```
// heap.h
// a binary min heap

#ifndef HEAP_H
#define HEAP_H

#include <iostream>

const int DEFAULT_SIZE = 100;

template <class KeyType>
class MinHeap
{
  public:
    MinHeap(int n = DEFAULT_SIZE);          // default constructor
    MinHeap(KeyType initA[], int n);        // construct heap from array
    MinHeap(const MinHeap<KeyType>& heap);  // copy constructor
    ~MinHeap();                             // destructor

    void heapSort(KeyType sorted[]);  // heapsort, return result in sorted

    MinHeap<KeyType>& operator=(const MinHeap<KeyType>& heap);  // assignment operator
    std::string toString() const;     // return string representation

  private:
    KeyType *A;     // array containing the heap
    int heapSize;   // size of the heap
    int capacity;   // size of A

          void heapify(int index);          // heapify subheap rooted at index
    void buildHeap();                 // build heap
        int leftChild(int index) { return 2 * index + 1; }  // return index of left child
        int rightChild(int index) { return 2 * index + 2; } // return index of right chil
d
        int parent(int index) { return (index - 1) / 2; }   // return index of parent
    void swap(int index1, int index2);        // swap elements in A
    void copy(const MinHeap<KeyType>& heap);  // copy heap to this heap
    void destroy();                           // deallocate heap
};

template <class KeyType>
std::ostream& operator<<(std::ostream& stream, const MinHeap<KeyType>& heap);

#include "heap.cpp"

#endif
```

```
// James Le - CS 271
// Feb 15, 2017
// heap.cpp
// C++ program for implementation of a MinHeap template class and Heap Sort algorithm

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <sstream>

#include "heap.h"
using namespace std;

/*================================================================
MinHeap Default Constructor
Pre: None
Post: Construct a new MinHeap data type with default size set to n
================================================================*/
template <class KeyType>
MinHeap<KeyType>::MinHeap(int n)
{
  A = new KeyType[capacity];
  heapSize = n;
}
/*================================================================================
MinHeap Array Constructor
Pre: An empty array initA with size n
Post: Construct a heap from array initA with contents from the heap from array A
================================================================================*/
template <class KeyType>
MinHeap<KeyType>::MinHeap(KeyType initA[], int n)
{
  A = new KeyType[capacity];
  heapSize = n;

  // Copy the array into the heap's internal array
  for (int i = 0; i < n; i++)
    A[i] = initA[i];

  // Organize the array into a proper MinHeap Tree
  buildHeap();
}

/*============================================================
MinHeap Copy Constructor
Pre: heap is a MinHeap data type
Post: Construct a MinHeap data type with same contents as heap
============================================================*/
template <class KeyType>
MinHeap<KeyType>::MinHeap(const MinHeap<KeyType>& heap)
{
  A = new KeyType[capacity];
  heapSize = heap.heapSize;

  // copy the array
  for (int i = 0; i < heap.heapSize; i++)
    A[i] = heap.A[i];
}

/*=========================================================
MinHeap Destructor
Pre: None
Post: Clean up the MinHeap class
=========================================================*/
template <class KeyType>
MinHeap<KeyType>::~MinHeap()
{
  if(A)
    delete A;
```

```
}

/*=================================================
Assignment Operator
Pre: heap is a MinHeap data type
Post: Allow assignment of values between MinHeap class
=================================================*/
template<class KeyType>
MinHeap<KeyType>& MinHeap<KeyType>::operator=(const MinHeap<KeyType>& heap)
{
  destroy();
  copy(heap);

  return *this;
}

/*================================
String Representation for MinHeap
================================*/
template<class KeyType>
std::string MinHeap<KeyType>::toString() const
{
  std::stringstream stream;
  stream << *this;
  return stream.str();
}

/*============================
Heapsort Algorithm
Pre: a list of integers
Post: sorted list of integers
============================*/
template <class KeyType>
void MinHeap<KeyType>::heapSort(KeyType sorted[])
{
  // Build heap (rearrange array)
  for (int i = heapSize/2 - 1; i >= 0; i--)
    heapify(i);

  // One by one extract an element from heap
  for (int i = heapSize - 1; i >= 0; i--)
  {
    // Move current root to the end
    swap(sorted[0], sorted[i]);

    // Call min-heapify on the reduced heap
    heapify(i);
  }
}

/*=============================================
Min-Heapify Subheap Rooted At Index
Pre: 2 subtrees for the children are already heaps
Post: Root A[0] is the root of the heap
=============================================*/
template <class KeyType>
void MinHeap<KeyType>::heapify(int index)
{
  int smallest = index; // Initialize smallest as root
  int l = leftChild(index);
  int r = rightChild(index);

  // If left child is smaller than root
  if (l <= heapSize && A[l] < A[smallest])
    smallest = l;

  // If right child is smaller than smallest so far
  if (r <= heapSize && A[r] < A[smallest])
    smallest = r;
```

```
  // If smallest is not root
  if (smallest != index)
  {
    swap(A[index], A[smallest]);

    // Recursively heapify the affected sub-tree
    heapify(smallest);
  }
}

/*============================================
Build MinHeap
Pre: None
Post: A heap that is sorted in decreasing order
============================================*/
template <class KeyType>
void MinHeap<KeyType>::buildHeap()
{
  heapSize = capacity;
  for (int i = (capacity/2); i >= 0; i--)
  {
    heapify(i);
  }
}

/*============================================
Swap Elements
Pre: 2 elements
Post: The 2 elements' index positions are swapped
============================================*/
template <class KeyType>
void MinHeap<KeyType>::swap(int index1, int index2)
{
  int temp = index1;
  index1 = index2;
  index2 = temp;
}

/*============================================================
Copy Heap to another Heap
Pre: heap is a MinHeap data type
Post: Construct a MinHeap data type with same contents as heap
============================================================*/
template <class KeyType>
void MinHeap<KeyType>::copy(const MinHeap<KeyType>& heap)
{
  A = new KeyType[capacity];
  heapSize = heap.heapSize;

  // copy the array
  for (int i = 0; i < heap.heapSize; i++)
    A[i] = heap.A[i];
}

/*============================================
Deallocate Heap
Pre: None
Post: Clean up the MinHeap class
============================================*/
template <class KeyType>
void MinHeap<KeyType>::destroy()
{
  if(A)
    delete A;
}
```

```
// test_heap.cpp

#include <iostream>
#include <cassert>
#include "heap.h"

using namespace std;

void test_heapSort()
{
  KeyType sorted[] = {12, 11, 3, 5, 6, 7};
  int heapSize = sizeof(sorted)/sizeof(sorted[0]);

  heapSort(sorted);
  assert(sorted.toString() == string("{3,5,6,7,11,12}"));
}

void test_heapify()
{
  KeyType A[] = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7};
  int heapSize = sizeof(A)/sizeof(A[0]);

  heapify(A[0]);
  assert(A.toString() == {1, 2, 3, 4, 7, 9, 10, 14, 8});
}

void test_buildHeap()
{
  KeyType A[] = {3, 8, 2, 1, 6, 5, 4, 7};
  int heapSize = sizeof(A)/sizeof(A[0]);

  buildHeap();
  assert(A.toString() == {8, 7, 3, 5, 6, 2, 4, 1});
}

void test_swap()
{
  KeyType A[] = {2, 3, 4, 5, 6, 7};
  swap(3, 6);
  assert(A.toString() == {2, 6, 4, 5, 3, 7});
}

void test_copy()
{
  KeyType A[] = {10, 6, 7, 14, 11};
  int heapSize = sizeof(A)/sizeof(A[0]);

  copy(heap);
  assert(heap.toString() == {10, 6, 7, 14, 11});
}

void test_destroy()
{
  KeyType A[] = {1, 2, 3, 4, 5, 6};
  destroy();
  assert(A.size() == 0);
}

int main()
{
  test_heapSort();
  test_heapify();
  test_swap();
  test_copy();
  test_destroy();

  return 0;
}
```